Poetic Software

Reading, Writing and Performing Software

(An exploratory essay about the in-between states of software
and how the hidden processes of computers are leaking to the surface)

Introduction

Software has taken command of our daily life.[1] It is omnipresent and most of our western society would come to a halt without it. At the same time software has become so ordinary, that it is often overlooked. Software is taken for granted while it is increasingly entangled in our life and constantly takes over new tasks. Our computers seem to become smarter through new kinds of algorithms. This leads to new challenges in understanding software – not only from a scientific point of view but also from a cultural, political and social perspective. And so software has also found its way into the art and vice versa, but there are still gaps in the relation between both of them. I think that the interaction between software and art can be productive and helpful for the research in both of the disciplines.

The questions that I am asking is: what can an artistic method for researching the processes and relations of software look like? ~~How can we look critically into the software that we use on a daily basis?~~ The current perception and use of software are significant parts of this research. Especially in contrast to the original culture around software, that included hacking and that required every artist to write their own software. This essay explores the multiple layers of software with a special focus on the dependencies and imaginations that arise around and through software. How are the entangled, hidden layers of software coming to the surface?

Software consists of several parts. One could divide it into the code, the compilation, the execution and its manifestation (e.g. visual output on displays or the computer reacting to mouse clicks). The code is a well researched topic and there have been many works that use code and programming for artistic purposes. The manifestation of software execution is what people are in contact with the most. At least partly. The visual outcome on the screen is what determines how user perceive software. But outcomes of software can also be invisible to the user, like data transmission, webservers or software for infrastructure. What is visible is mostly not the software itself but its result of execution (a webpage or trains going back and forth). The most abstract part is the execution, but at the same time it is the most crucial part of software. During runtime machinecode[2] turns into machine commands and into physical current, resulting for instance in a change of pixel colours. This complex interplay, when the code turns into machine action is already in itself an act of poetic expression – an interpretation of the code through the machine, an in-between state with clearly defined rhythmic. The exact moments of this transitions are beyond human perception. So a division of software would only simplify the complex inter-dependencies the different parts have. It is exactly this moments of transition, the in-between states, the dependencies, that this essay tries to emphasize.

During the research of this project, I found myself returning to the essay “There is no Software” by Kittler over and over again, drawing inspiration from and following up on the various issues touched upon by Kittler. I uncovered a great variety of controversies surrounding the creation, execution and use of software. Furthermore, I realized that the more research I did on software and its implications for our lives the more aware I became of the software that I have been using. I started observing my own attitude towards various applications that have been shaping my life and work

1 Referring to the book of Lev Manovich, “Software takes command”

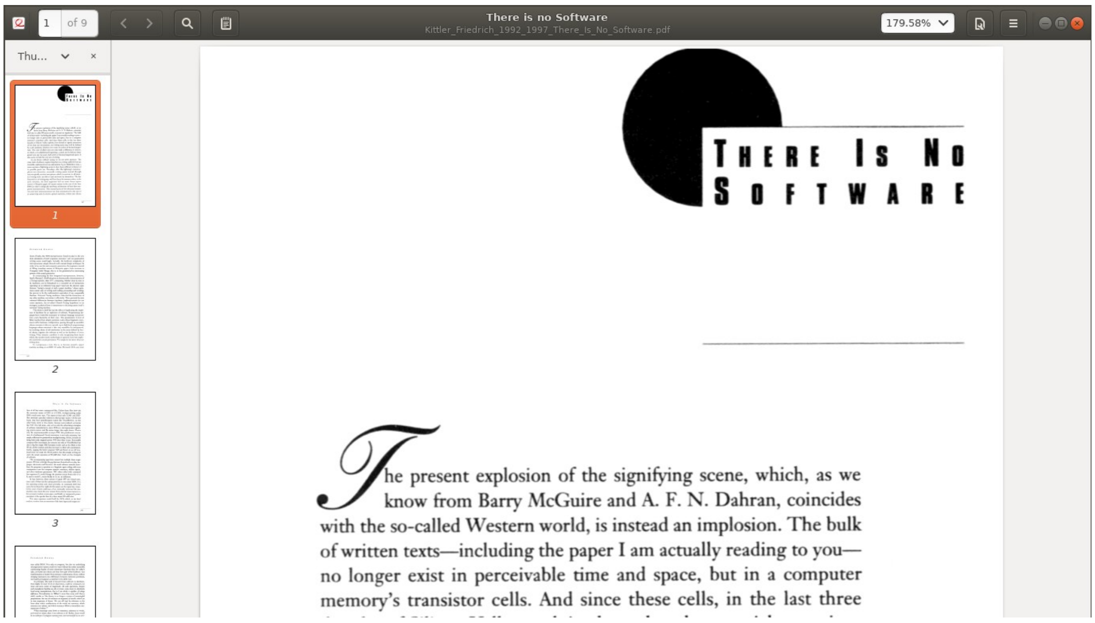
2 The human readable code is transformed into machine code through compilation. It is a process of translation from human readable instructions to machine instructions. Only the machine code can be executed by the machine, so the part of compilation is crucial to the creation of software.

everyday and started questioning many functions and backgrounds of software that I had viewed as a given before.
I became an Ethnographer of my own work in progress. I realized that my own behaviour and everyday occurrences in the interaction with software reflected what I was reading in research papers and articles on my screen and vice versa. Kittler therefore serves as a point of departure for different controversies around software. This will also lead me to the arts, and why I think art might provide possible approaches towards these different topics.

The method
The first part of my work will be an ethnographically inspired examination of the interaction with my computer whilst reading Kittler's essay »There is no software«. The text will unfold on two different levels: On the one hand I am describing the process of reading while interacting with the software I use to do so. On the other hand there will be interventions to critically reflect on various concepts touched upon. These interventions refer to either Kittler's text itself, or to the software that I am using. In the second part I will describe how art provides different frameworks to approach the different aspects I pointed out in the first part and how art and software relate to each others practices.

Why this method?
The detailed description of reading digitally makes the different software that is being used visible. Through that the software can be observed while at work. Next to this it is a great chance to revisit the text of Kittler. This method also allows for new encounters and associations with software, that will help to recognise the different agents at stake when thinking about the processes of software and the involvement of art with it.

Why »There is no Software« by Kittler?
This text very early became one of the key texts of my interest and research. The text offers a great source for thinking about software today. Because in his essay from 1992 he is actually not negating the existence of software, instead he wants to emphasize the materiality, that is being neglected in his opinion. This is a huge tension that we can also recognize in computation today. Even if we do not neglect software, it becomes more and more invisible, we imagine software mostly through metaphors. Workflows are so seamless it seems almost like *there is no software.*



THERE IS NO SOFTWARE

The present explosion of the signifying scene, which, as we know from Barry McGuire and A. F. N. Dahran, coincides with the so-called Western world, is instead an implosion. The bulk of written texts—including the paper I am actually reading to you—no longer exist in perceivable time and space, but in a computer memory's transistor cells. And since these cells, in the last three

Illustration 1: The document viewer showing "There is no Software" by Kittler

1 I am reading, the computer is reading

or how to observe software

I am reading “There is no software” by Friedrich Kittler. I downloaded the pdf file of the text to my computer using the Firefox browser. The browser has saved the file in my downloads folder. I can find it through the file-system, which I can view in a representational view by opening the file explorer. By clicking the icon of the file explorer the computer opens a new window for me. I see different icons of folders and many other ones for files. I double click my way through the folders until I end up in the Downloads folder, where the newly downloaded file is placed in a list view among others. The file is called “Kittler_Friedrich_1992_1997_There_Is_No_Software.pdf”. I hover the small bar with the title. The operating system default setting is to open the file with the document viewer and so I do, by double clicking the left mouse button. Within seconds a new window appears putting the file manager window into the background and foregrounding the title page of the pdf framed by small icons and scroll-bars. I click to enlarge to full-screen and start to scroll down till the first lines of text appear. I zoom out pressing the combination CTRL and - twice. Next, I start reading the first sentence. “The present explosion of the signifying scene, which, as we know from Barry McGuire and A F. N. Dahran, coincides with the so-called Western world, is instead an implosion.” Barry McGuire? I hover the name, press the mouse down and drag from B to e. The text tints white with a blue background. The release of the mouse button is followed by pressing CTRL C. I switch to the browser, which still shows the download page of the PDF. I paste the name into the search bar and press enter. The search engine shows a list of results – one video, this must be it. As the link reacts to my hovering, I click on it and with a short flickering I end up on YouTube. Without any action required the video starts and the speakers play: “The eastern world it is exploding”, to which Kittler must have referred.

Compression

The implosion and explosion can well be seen on different levels of software. While the complexity and interplay of different technologies are exploding, the visibility and the potential for understanding are imploding. Increasingly better software brings great advances in e.g. computer vision, but at the same time it becomes harder to understand. The potential of having more sophisticated technology may come at the risk of blurring the understanding. At the same time these highly complex algorithms require more hardware and even better processors.

Illustration 2: Logo of WinRar

The implosion of files is a very well used method in the form of compression. Compression needs software that is able to rearrange the bytes of files using various algorithms, for the sake of file size. Smaller files can be stored easier and have advantages for transmitting. But this can have different implications. It is a method to circumvent the physical limitations (to some extend). This means that files can be stored with very little storage available.

Other than that, we produce increasingly bigger files, because cameras output high-resolution images, we can gather more data, scan better and display highly sophisticated websites. But how directly does this effect us? Unlike the imagination that the digital is immaterial, the processing of big files for instance is consuming much energy (De Decker 2018). Therefore some websites like the lowtechmagazine are developing different methods on how to host low-energy web pages. They are using solar panels and produce their websites in a way that makes the site very light in terms of data that has to be transmitted. So from this case we can see, that compression can have multiple effects. It is the small nuances that make software a powerful tool to think about current cultural topics. This lightweight approach gives reason to think about different aspects of how websites are being served and how they are built.

Space is a recurrent scheme in computation. Computer Science tries to shrink and expand at the same time. It is almost like a play that can be occupied on different layers. The »compression of space« onto the size of a micro chip is opposed with the exploding need for power or, to remain within the metaphor of space, disk space (Kitchin 2011, p. ix). The expansion of the digital does not remain within the computer, but it is actively becoming part of our real spaces. »[S]oftware generates behaviors and opportunities, and traffics in meanings, readings, and interpretations« (Kitchin 2011, p. ix). To figure out these exact moments of influence and these borders between computers and the real world might be very hard to accomplish, if not impossible.
As Hu points out, the material and the digital world are interwoven more deeply than we think. For example, the imagination of the internet as a *cloud* manifests in the real world as cables that get placed across oceans and buildings that hold thousands of servers (Hu 2016, p. 6).

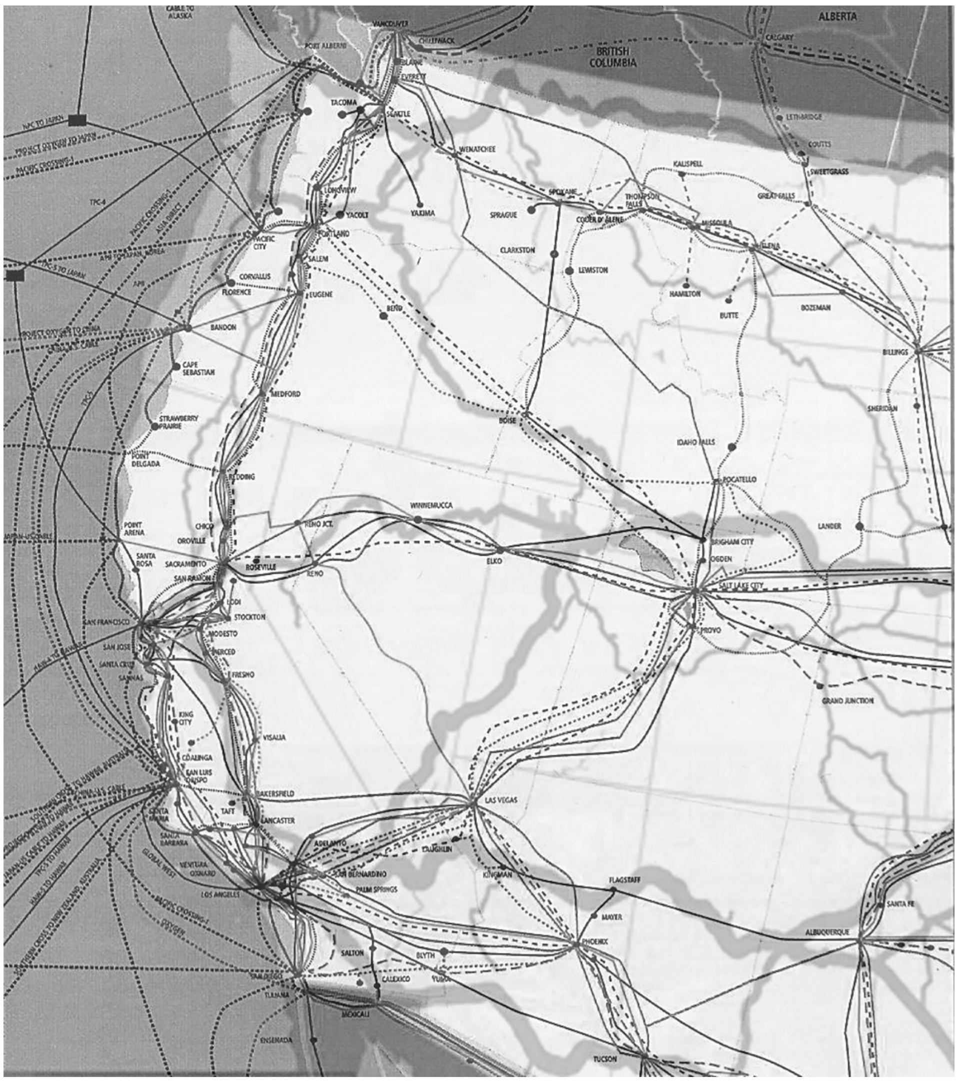


Illustration 3: the internet in its physicality. Cables are following old railroad routes. (add source to prehistory of cloud here) (Hu 2016, p. 3)

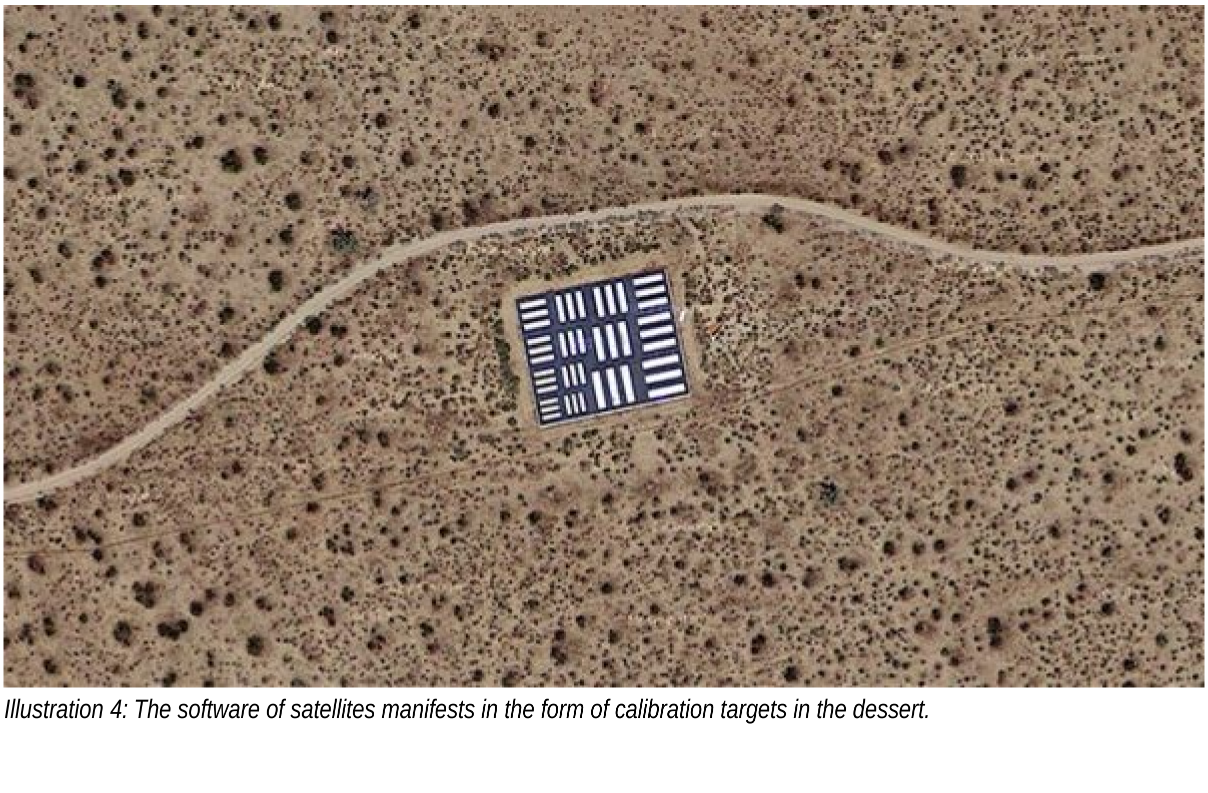


Illustration 4: The software of satellites manifests in the form of calibration targets in the dessert.

I stop the video by clicking onto the face of the singer and a smoothly appearing pause sign inside a circle signals the success of my action. I change back to the document viewer by clicking on the window that got hidden in the background by the browser.
The words I read are displayed with a grained border presumably caused by the scanning process. As I read on, my t-shaped cursor follows the lines of the text. I continue with the next sentence. »The last historical act of writing may well have been the moment when, in the early seventies, the Intel engineers laid out some dozen square meters of blueprint paper« (Kittler 1992).

Dependencies
With its increasing speed, computation fosters itself while depending on the previous version of its own. The same holds true for software. Therefore we can recognize a spiral of dependencies and influences that includes humans and machines. After the first hardware was able to draw new, even smaller hardware than it would ever have been possible with paper and pen, the system of hardware design became dependent on itself (Kittler 1992). This means the next generation of hardware is always enabled by and relying on the previous version, making it possible to create even smaller and more complicated parts. The same can be found in the culture of software development. Software can only be built with software: software that enables to write the program code, software that compiles the code into machine readable binary-code and an operating system that executes it. This also means that nearly every program relies on other ones, requiring users to pre-install specific versions of software in order to run the program. If one single component of this chain of dependencies breaks, many other programs will be affected.
The dependence on companies that produce software is great. If the company decides to discontinue their software, the user is directly influenced by it as he can not use his software anymore. This happened for instance, when Microsoft shut down one of their scripting languages, many companies that relied on it suffered (Ullman 2012, p. 105).
Software is changing over time. As Ellen Ullman mentions, this is also the reason why software can quickly become unstable, especially if multiple programmers are working on a program over a longer lasting period of time. Code can be written in many ways different ways making it hard for other persons to read or understand. Still, huge systems with a long history have to be kept running as many other systems rely on them making it a very fragile system, in which you can hardly change anything (Ullman 2012, p. 117). These kind of dependencies tell their own stories and are rarely clearly visible. Also these dependencies have the potential to show the history of software. It clearly shows that software can not exist on its own, but is always embedded into a greater ecosystem, a cultural framework that follows its own rules.

»We shape our tools and, thereafter, our tools shape us« (Davis 2016) says a famous quote by John Culkin from 1967. But if we look at the dependencies of software one could also say: we shape tools and these tools shape new tools again. Transferring this idea to the notion of software as a cultural object, the interrelation between shaping and being shaped could be formulated as follows: software creates and influences culture, and therefore this culture shapes new social conditions under which the construction and use of software itself is altered. This might become clear when looking at the example of software-hacking. The distribution of proprietary software with Digital Rights Management (DRM) lead to multiple groups cracking and circumventing software limitations. These cracks are then distributed as new software.

The original culture of software was actually build around open source culture, even though this concept wasn’t formulated yet – because it was not necessary. Early software production was very dependent on this openness. (Mansoux 2017, p. 31). Without the sharing of software and code, the development would have been very tedious, if possible at all.

I further follow the dark pixels on the screen to the roaring sound of the computer. It is not clear whether the ventilation sound is triggered by the hardware or the software, which is causing the CPU to overheat. Kittler is writing about how the language gets abstracted from high-level, human readable words, to assembler code, that is being translated into non readable machine code. As Kittler talks about this »postmodern Tower of Babel« (1992, p. 148) I realize how my windows have started to built up like a tower. The document viewer on top of the browser on top of the settings on top of the mail program and so on.

Framework culture

Programming languages are based on other programming languages in order to make the code easier to write and read. Low-level languages are very close to the actual machine processes and therefore very complex to write. This is why high-level languages were constructed to translate this elaborate processes into human readable concepts and language. In addition to that programmers often rely on third party frameworks, which provide functions that are very convenient to implement. Instead of having to write the code themselves, they just have to put one line of import. Therefore the whole set of tools provided by the so called library becomes available for the use of the programmer. The process of using frameworks often obscures the actual algorithms. For example it can be quite challenging to create a machine learning algorithm from scratch but frameworks like »keras« or »tensorflow« make it accessible. The problem is that the programming syntax is very close to human language, which makes it hard to comprehend the actual code. Thus it is harder to change functions that are underneath the layer of the framework-interface (Cox 2007, p. 153).

Furthermore the different programming languages favor different concepts of language and writing as well (Cox 2007, p. 153). So the choice of programming language already determines a certain style of writing. And, because language significantly shapes our imagination, the choice of programming language also influences our understanding of software. Although scripting languages are very popular right now, it can not replace low-level programming at all.

»High-level programming approaches can be very successful in achieving certain ends, but the very imposition of higher-level constructs and metaphors also limits awareness of how code operates in and for itself and what may be achieved through that. Arguably it is the changes in low-level systems that have provoked the biggest paradigm shifts, such as the development of binary computation and Turing machines [...]« (Yuill 2004)

To me this also means that an active engagement with different levels of programming is necessary to reflect important aspects of computation. A critical practice around software should therefore not only focus on one specific programming language. This helps to free yourself from the dependencies stated above and enables you to engage on different layers, not only the surface.

I continue in the text, and while Kittler is buying a commercial version of WordPerfect, I remember my old copy of Word that is still installed on my old partition. I go through the folders of my applications folder of my second partition scan through all the apps, that I probably haven't used for months. I follow the alphabetical order of the list view and after some programs starting with "N", appears a folder called Microsoft. I double click on the icon of an orange folder and end up in a grid view, containing 6 files and some folders. In-between them: word.exe. The executable file to open Word. I can't execute it on Linux.

I am a consumer not a user

Nowadays the software that is required to use a machine comes pre-installed and ready to use. Software can be downloaded from centralized marketplaces: App Stores. This causes an immense dependence on the producers, that are again depending on owners of these marketplaces (including their platform framework and policies). These producers have developed an infinite selection of apps for everything. This is another example for the »explosion« of software that was previously mentioned. This flood of applications causes software to become a mundane occurrence. The danger of that is that we take software for granted. When we have a problem, there is an app for it. Nobody thinks about the possibility of editing software and adjusting it to one's need. This is not only because most of the time it is not possible to edit the software due to DRM but also because the average user is not a user anymore. Rather people are being educated by companies to be consumers instead of users let alone creators.[3] It is in the companies interest to make their clients dependent on their product. Therefore companies are not interested in opening up their products, but they are instead locking it up. They are then slowly feeding their clients with updates and new fancy features. This is great for users who just need to get their job done and who want to be in contact with technical struggles as little as possible. On the other hand it means that firstly, the use of software is dictated by companies and secondly that IF you want to engage with your software you can't do so. You can't look at the source code, reuse parts of it and you can't modify the program to your needs.

Of course there is also another end of the spectrum: hackers and creators with custom software and completely refusing any use of commercial software. This movement also provides a great source for discussion about software. The problem is that the average user is not happy about struggling to install what they need before they can actually write something. There are also other kinds of software, that embrace the user as an active agent, while still enabling an easy use on the surface. For example the mediawiki software allows for easy editing on the browser, while still providing an infrastructure to easily extend the functions.

»The accompanying paperware« – which paperware? Where is the manual of my document viewer? I move my mouse towards the options on top of the window and click on help. A small window opens, displaying a table of contents. »How to use it« »Find text in documents«… A page con-

3 The definition of user has changed through the years. In the beginning of computation there was basically no distinction between a user and a programmer, because of the simple fact, that users had to program.

taining hyperlinks for different sections. It is probably the first time I ever entered this space of the program.

Tell me what to do

Software can be so abstract, that the way how software affects people is often through the metaphors it uses. What we remember is the animal on the start-screen, not the algorithm that it uses. For an artistic engagement I think it is important, to carefully examine the different parts of software and then reflect on their use – like the metaphor of the user manual.
The manual of most programs is part of the software. Actually, the manual is software. The handbook does not come in a physical form anymore. Just as the software does not ship on Floppy or CD-ROMs. Software is a download (or a service, that is only running online[4]), so it never really enters the physical space anymore and thus, it becomes even more abstract. Through the handbook, the software manifests itself as a tool. A tool, that has certain functions and the manual describes how to use those functions correctly. Nowadays, the handbook often constitutes a space that stays undiscovered. If we want to consider software as an artistic material, the handbook can also gain new functions as a description, as a space for thoughts. The handbook was also used as a metaphor at the readme festival 2005 to guide visitors through an exhibition of software. Software often remains invisible in its functions and statements, so it is necessary to describe what it is doing. The manual illustrates the fact, that the "user" needs to be informed what to do with software and how to use it.

I close the help, and find my way back to the text. In the meantime, Kittler turns towards his punchline: There is no software.

Software is conceptual

Even though software is depended on hardware, it does not mean that there is no software. A deeper engagement with software also means taking software seriously. Even though it might be argued that software is only the representation of machine operations, it is important to acknowledge software as an independent object of study.
Even though Kittler was arguing that there is no software and it is intrinsically connected to its hardware, Cramer points out that »if any algorithm can be executed mentally, as it was common before computers were invented, then of course software can exist and run without hardware« (Cramer 2002a). Following this argument it points to the idea of software in a very conceptual way, not only defining software as a program that is running on a certain hardware. All layers of diminishing abstraction on top of hardware deserve attention. Still it is important to recognize both of the perspectives for their importance – the materialistic and the cultural / political.
Anyway, there is no clear border between software and hardware. Where does software begin and Hardware end? Is it when the Code is being compiled or is it when the machine code is transformed into electrical signals? Eventually, the exact point where the Software transforms into Hardware is not clearly perceivable (Tenen 2017, p. 88).

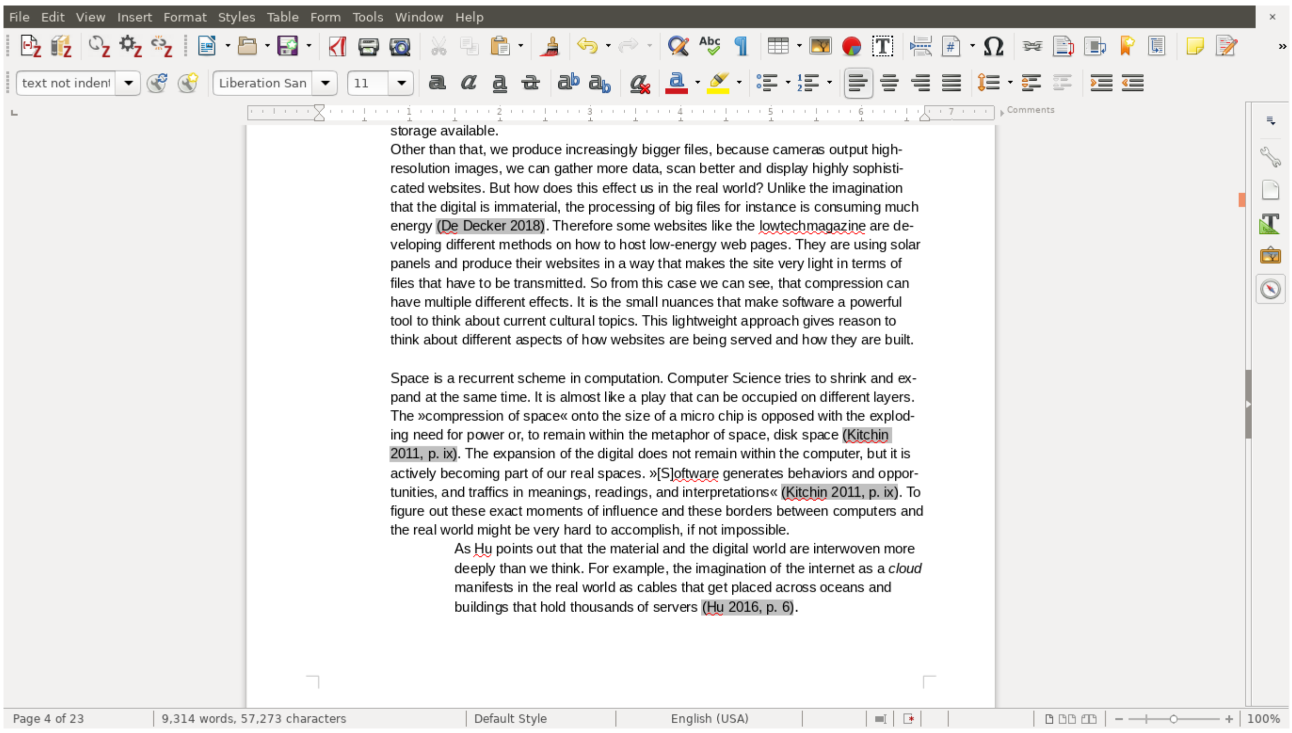
4 The concept of software as a service (SaaS) is a very current issue in software. The software is not running on the computer of the user but rather on the server of the provider. This means that the user does need an internet connection and is constantly sending data to the server. In addition the user is not in hold of any executable file or program anymore, ending up in even more issues around dependencies.

There is certainly a tension between the development of software and hardware. The hardware limits the software. We can not build applications that run faster than the hardware. Machine Learning algorithms for example need a lot of resources to calculate their models. This means that effective research with this technology is only possible with sufficient hardware. Even though software can be seen as a conceptual good, it is impossible to execute it only mentally, especially when using very complicated algorithms. Software is only effective through its execution, its performance.

I continue with reading Kittler. »First, on an intentionally superficial level, perfect graphic user interfaces, since they dispense with writing itself, hide a whole machine from its users.« (Kittler 1992, p. 149)

The graphical user interface
The user interface enables a convenient way to display software (or at least parts of it). This representation is however only an interpretation of what the designer thought is the best way to display it (Hadler et al. 2016, p. 7). At the same time it looks like this user interface is the only truth that the program holds. It does certainly not become obvious that this interface is not neutral. The GUI instead hides. It hides the processes, a lot of functions, the source code, the possibilities, the decision it takes for you.

The need for a human approach to software also becomes visible from the great use of Graphical User Interfaces. The so called GUI, is not part of the original imaginary of computation, where commands were being filled in via a command line. But today’s average user is only surrounded by software displayed via a “window”, encountering the terminal only by chance. Not only does the GUI simplify commands into buttons and mouse-actions, but also does it make software more human. A button that has a 3D effect, the on/off function is displayed via a switch, the mouse transforms into a hand or the form that looks like a letter, which off course you fill in by pressing a pen symbol (Fuller 2008, p. 175). This is also known as skeuomorphism. It means that objects of the real world are being used for representing digital functions or interface objects. Humans anthropomorphize and use metaphors to communicate the complexities of a less well known domain (the digital) via the vocabulary and concepts associated to a well known domain (the physical world). The skeuomorphism in GUIs is a good example of that.


File Edit View Insert Format Styles Table Form Tools Window Help
text not indent
Liberation San
11
Comments
Page 4 of 23
9,314 words, 57,273 characters
Default Style
English (USA)
100%

storage available.
Other than that, we produce increasingly bigger files, because cameras output high-resolution images, we can gather more data, scan better and display highly sophisticated websites. But how does this effect us in the real world? Unlike the imagination that the digital is immaterial, the processing of big files for instance is consuming much energy (De Decker 2018). Therefore some websites like the lowtechmagazine are developing different methods on how to host low-energy web pages. They are using solar panels and produce their websites in a way that makes the site very light in terms of files that have to be transmitted. So from this case we can see, that compression can have multiple different effects. It is the small nuances that make software a powerful tool to think about current cultural topics. This lightweight approach gives reason to think about different aspects of how websites are being served and how they are built.

Space is a recurrent scheme in computation. Computer Science tries to shrink and expand at the same time. It is almost like a play that can be occupied on different layers. The »compression of space« onto the size of a micro chip is opposed with the exploding need for power or, to remain within the metaphor of space, disk space (Kitchin 2011, p. ix). The expansion of the digital does not remain within the computer, but it is actively becoming part of our real spaces. »[S]oftware generates behaviors and opportunities, and traffics in meanings, readings, and interpretations« (Kitchin 2011, p. ix). To figure out these exact moments of influence and these borders between computers and the real world might be very hard to accomplish, if not impossible.

As Hu points out that the material and the digital world are interwoven more deeply than we think. For example, the imagination of the internet as a *cloud* manifests in the real world as cables that get placed across oceans and buildings that hold thousands of servers (Hu 2016, p. 6).

Illustration 5: The writing of this essay with LibreOffice

As I go further in Kittlers text, focusing on the text as my mail software wants to interrupt me with some notifications about incoming mails. I click them away. Kittler is writing about how computers are writing and reading themselves. I want to copy this part into my notes. I drag the mouse from »in contrast« to »read and write by themselves« and as the text tints, the layer of text reads: »in contnast to all histor- ical writigtools, are able to read and write by thenvselves« (sic! by ocr?) (1992, p. 147). My machine has read the text before me – not only once. Actually the text has probably been written and read many times before I opened it. The computer had read the document for words using Optical Character Recognition and even made its own interpretation. That explains why the selected text is wrong, because the program misinterpreted some of the characters. Together with this not incorrect version of the text, it got written again to the memory. Then another time the text was read – into the working memory, when I opened it with the document viewer.

Glitches and non-functional software

We can get a spark of what execution of code means and how software really acts and performs when it fails or when it is taken out of its context (Winograd and Flores 1995). So in the following I want to show that for a serious engagement with software it is also necessary to look at the non-functional and the stuff that is in-between the pixels and conducting paths. Software is primary made to function, but what if software fails or malfunctions? What, if software has no function?

“Most people notice infrastructures only when they are put in the wrong place or break down. This means that public knowledge of them is largely limited to their misplacement or malfunction.” (Parks 2009)

While The Alliance for Code Excellence imagines »[a] world where software runs cleanly and correctly as it simplifies, enhances and enriches our everyday life is achievable« (Constant 2018, p. 11) I argue that the malfunctioning of code can also be something positive that is revealing and holds a value. The interruption of a seamless flow, makes undeniable apparent, what couldn't be seen before. We can use things without being immediately aware of their presence, but the "breakdown" makes them visible. So the malfunctions "reveal[e] to us the nature of our practices and equipment, making them 'present-to-hand' to us, perhaps for the first time." (Winograd and Flores 1995, pp. 77–78)
For example the wrong character recognition as visible from the text above, can also show how the algorithm works. The mistaken "m" for "rn" shows that the algorithm might work with visual comparison and has probably not recognised the gap between "r" and "n" – due to the grain of the text. This consequently gives a clue, that the algorithm doesn't have an idea about the context of words, otherwise it would have figured out that some words are not correct English words.
The way how software is set up, can embrace the fact that software is failing or not. In the case of seamless software that tries to hide failure the user does not get any insight. In contrast, when the setup is embracing its unstable character, the user knows that there is a potential for crashes. It means that engagement is undeniable. At the point when it crashes you will be able to get a glimpse of the inner workings of software and possibly be able to even fix it.

The imperfection of software
Digital System are often considered to be pixel-perfect. But instead also digital applications become unstable. This is also a result of the dependencies and glitches as pointed out before. Software can even have the same noise as non-digital objects have. When Casey Reas wrote about the new Processing[5] he pointed out the high precision that computers have compared to similar art-forms like Sol LeWitt practiced it[6]. »[…] [M]achines can draw lines with absolute precision so all the imperfections in a physical drawing are removed, giving the rendering different characteristics than those intended by LeWitt.« (Reas 2019) In reality it turned out that after a few month processing produced the same inaccuracies (glitches) as a drawing by LeWitt would show. This was due to updates and changes in the language.

5 A software framework to make programming for artists more accessible
6 Sol LeWitt was famously known for giving painters instructions on how to draw paintings. His work is also often used as a reference for digital art that follows formal instructions, just like software does for instance.

Illustration 6: Software error at McDonalds Regensburg, Germany

I change from the document view into the writing program LibreOffice, where I store most of my notes. With a single click on the icon, no keystroke required, the execution starts and the start screen appears. Many process get triggered by this simple action and the computer follows its instructions, which I do not know – and not even see. But not with ease this time. The only thing that I can occupy right now, that the process must have »frozen«. As my mouse indicates with a spinning motion, I am unable to continue. I am unable to change the program, I am stuck, just like my program. I try clicking on the icon, again and again, as if my actions would trigger the program to finally make it. It is as if I want to tell the program to try harder by clicking harder. Once again I try to encourage the app, by clicking somewhere randomly on the screen. I give up. I have had this before, so I know how to act. »kill«[7]. I change to the terminal, type »sudo killall libreoffice«. I give my permission by typing in my password and happily I can see the terminal taking action. With a flicker the startup screen that was stuck disappears, freeing me and my cursor from redundant spinning. I try restarting the program and hope, that the crash was only due to unlucky circumstances, maybe just something »got stuck«.

Imaginative Software

The perception of software is anything but neutral. Software tells stories, through its metaphors, its contents, its performance. The digital medium offers new ways of telling stories. This becomes obvious not only due to different structures, like the form of the database as Lev Manovich points out, but also because of the different modes of intervention software takes in our life (Manovich 1999, p. 82). The medium keeps evolving at inexorable speed and

7 Kill is a command in unix, that sends a signal to quit processes. (Anon. 2018)

so does software, leaving space for new ways of how to tell and what to tell about computation.

That humans tend to anthropomorphize not only their surroundings but also computers and technology in general has been a well researched topic among computer sciences & psychology. Among others, “The media equation” had shown, that we as humans consciously and unconsciously anthropomorphize computers (Reeves and Nass 2003). In addition to that humans have a vivid and diverse imagination about processes that are invisible. This includes software. Often digital media black-boxes certain processes and therefore provides a lot of space for imagination and narratives that can be constructed around it (Finn 2017, p. 229). Narratives have been used for the purpose of marketing and there have been attempts to create relatable stories within applications. A well known example is Joseph Weizenbaum’s Eliza, a digital application, that acted as a therapist, chatting with the user. This piece of software gave impressive proof of how humans anthropomorphize even simple digital applications (Wardrip-Fruin 2012, p. 27). Tech giants have put great effort in implementing relatable characters into their systems, e. g. voice assistants. An assistant, that is helpful and funny, that gathers you data with great pleasure. But in the past there have also been unsuccessful attempts to add anthropomorphizing elements to programs, only to remind quickly about Microsoft’s famous Clippy (Cain 2017). These stories in applications and around it make technology more understandable, but can also be source for misconceptions. A current example seems to be the fear of singularity after machine learning enables applications to »magically« generate or label images. The gap between the real potential and the imagination about it is big. I don’t want to support an uncritical or blind approach towards technology – I think it is important to be realistic, critical and playful equally with these algorithms, only then turns engagement into insight.

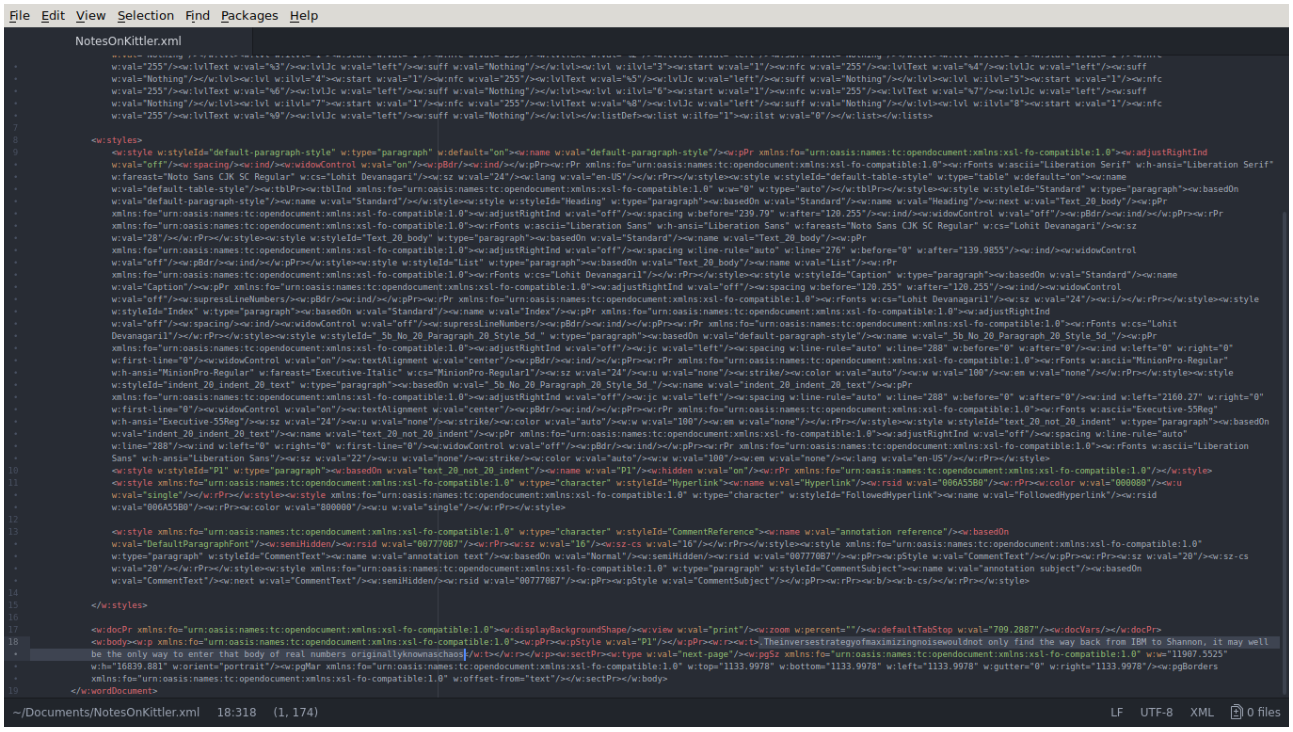
Another case of narratives is the narrative that exists outside the software. It lies in its performance. How it acts, where and when. The realization that people relate to software on an emotional level, makes it possible to create software that tells more than its function. Actually it’s possible to tell stories only by how software works. This kind of narrative has been used in some works of Software Art. For example a work by Luca Bertini which can be found on run-me.org. The work is about two viruses in love. »They search for each other on the net, running through connected computers« (Bertini 2019).

I restart LibreOffice – this time it works. An empty document opens, and a blinking cursor indicates, that I am ready to type. I switch back to the text viewer where Kittlers text is waiting for me and I copy the last sentence. After clicking my way back into my editor I paste the string form the clipboard to my empty document. Immediately the text fills the screen:
».Theinversestrategyofmaximizingnoisewouldnot only find the way back from IBM to Shannon, it may well be the only way to enter that body of real numbers originallyknownaschaos«. My computer completely rewrote the original text.

Noise in Software

The polished interface makes us forget about what programmers struggle with every day. The noise that surrounds computation. It is the same noise that should make us aware of how imperfect and subjective software is, but in many cases this noise is being suppressed. Every small glitch is being removed out of software and every irregularity is considered as a bug. But this noise might instead be the possibility to further explore new opportunities with Code and its execution. Maybe the beauty of software lies in exactly this noise, that is being forgotten about in-between the logical operations with 0s and 1s.

I save the file and the machine once again writes for me to the hard-drive. I store it using the file-format xml. The file gets stored using the name NotesOnKittler.xml into the Documents folder. If I open the text in a normal text-editor it turns out, the computer has actually written noise around the actual text that I saved. This noise makes up the standards of the .xml format, encoding information within <tags>.



```
        w:val="255"/><w:lvlText w:val="%3"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="3"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%4"/><w:lvlJc w:val="left"/><w:suff
        w:val="Nothing"/></w:lvl><w:lvl w:ilvl="4"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%5"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="5"><w:start w:val="1"/><w:nfc
        w:val="255"/><w:lvlText w:val="%6"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="6"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%7"/><w:lvlJc w:val="left"/><w:suff
        w:val="Nothing"/></w:lvl><w:lvl w:ilvl="7"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%8"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="8"><w:start w:val="1"/><w:nfc
        w:val="255"/><w:lvlText w:val="%9"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl></w:listDef><w:list w:ilfo="1"><w:ilst w:val="0"/></w:list></w:lists>

    <w:styles>
        <w:style w:styleId="default-paragraph-style" w:type="paragraph" w:default="on"><w:name w:val="default-paragraph-style"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd
        w:val="off"/><w:spacing/><w:ind/><w:widowControl w:val="on"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Liberation Serif" w:h-ansi="Liberation Serif"
        w:fareast="Noto Sans CJK SC Regular" w:cs="Lohit Devanagari"/><w:sz w:val="24"/><w:lang w:val="en-US"/></w:rPr></w:style><w:style w:styleId="default-table-style" w:type="table" w:default="on"><w:name
        w:val="default-table-style"/><w:tblPr><w:tblInd xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:w="0" w:type="auto"/></w:tblPr></w:style><w:style w:styleId="Standard" w:type="paragraph"><w:basedOn
        w:val="default-paragraph-style"/><w:name w:val="Standard"/></w:style><w:style w:styleId="Heading" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name w:val="Heading"/><w:next w:val="Text_20_body"/><w:pPr
        xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:before="239.79" w:after="120.255"/><w:ind/><w:widowControl w:val="off"/><w:pBdr/><w:ind/></w:pPr><w:rPr
        xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Liberation Sans" w:h-ansi="Liberation Sans" w:fareast="Noto Sans CJK SC Regular" w:cs="Lohit Devanagari"/><w:sz
        w:val="28"/></w:rPr></w:style><w:style w:styleId="Text_20_body" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name w:val="Text_20_body"/><w:pPr
        xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:line-rule="auto" w:line="276" w:before="0" w:after="139.9855"/><w:ind/><w:widowControl
        w:val="off"/><w:pBdr/><w:ind/></w:pPr></w:style><w:style w:styleId="List" w:type="paragraph"><w:basedOn w:val="Text_20_body"/><w:name w:val="List"/><w:rPr
        xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:cs="Lohit Devanagaril"/></w:rPr></w:style><w:style w:styleId="Caption" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name
        w:val="Caption"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:before="120.255" w:after="120.255"/><w:ind/><w:widowControl
        w:val="off"/><w:supressLineNumbers/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:cs="Lohit Devanagaril"/><w:sz w:val="24"/><w:i/></w:rPr></w:style><w:style
        w:styleId="Index" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name w:val="Index"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd
        w:val="off"/><w:spacing/><w:ind/><w:widowControl w:val="off"/><w:supressLineNumbers/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:cs="Lohit
        Devanagaril"/></w:rPr></w:style><w:style w:styleId="_5b_No_20_Paragraph_20_Style_5d_" w:type="paragraph"><w:basedOn w:val="default-paragraph-style"/><w:name w:val="_5b_No_20_Paragraph_20_Style_5d_"/><w:pPr
        xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:jc w:val="left"/><w:spacing w:line-rule="auto" w:line="288" w:before="0" w:after="0"/><w:ind w:left="0" w:right="0"
        w:first-line="0"/><w:widowControl w:val="on"/><w:textAlignment w:val="center"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="MinionPro-Regular"
        w:h-ansi="MinionPro-Regular" w:fareast="Executive-Italic" w:cs="MinionPro-Regularl"/><w:sz w:val="24"/><w:u w:val="none"/><w:strike/><w:color w:val="auto"/><w:w w:val="100"/><w:em w:val="none"/></w:rPr></w:style><w:style
        w:styleId="indent_20_indent_20_text" w:type="paragraph"><w:basedOn w:val="_5b_No_20_Paragraph_20_Style_5d_"/><w:name w:val="indent_20_indent_20_text"/><w:pPr
        xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:jc w:val="left"/><w:spacing w:line-rule="auto" w:line="288" w:before="0" w:after="0"/><w:ind w:left="2160.27" w:right="0"
        w:first-line="0"/><w:widowControl w:val="on"/><w:textAlignment w:val="center"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Executive-55Reg"
        w:h-ansi="Executive-55Reg"/><w:sz w:val="24"/><w:u w:val="none"/><w:strike/><w:color w:val="auto"/><w:w w:val="100"/><w:em w:val="none"/></w:rPr></w:style><w:style w:styleId="text_20_not_20_indent" w:type="paragraph"><w:basedOn
        w:val="indent_20_indent_20_text"/><w:name w:val="text_20_not_20_indent"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:line-rule="auto"
        w:line="288"/><w:ind w:left="0" w:right="0" w:first-line="0"/><w:widowControl w:val="off"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Liberation
        Sans" w:h-ansi="Liberation Sans"/><w:sz w:val="22"/><w:u w:val="none"/><w:strike/><w:color w:val="auto"/><w:w w:val="100"/><w:em w:val="none"/><w:lang w:val="en-US"/></w:rPr></w:style>
        <w:style w:styleId="P1" w:type="paragraph"><w:basedOn w:val="text_20_not_20_indent"/><w:name w:val="P1"/><w:hidden w:val="on"/><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"/></w:style>
        <w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="character" w:styleId="Hyperlink"><w:name w:val="Hyperlink"/><w:rsid w:val="006A55B0"/><w:rPr><w:color w:val="000080"/><w:u
        w:val="single"/></w:rPr></w:style><w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="character" w:styleId="FollowedHyperlink"><w:name w:val="FollowedHyperlink"/><w:rsid
        w:val="006A55B0"/><w:rPr><w:color w:val="800000"/><w:u w:val="single"/></w:rPr></w:style>

        <w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="character" w:styleId="CommentReference"><w:name w:val="annotation reference"/><w:basedOn
        w:val="DefaultParagraphFont"/><w:semiHidden/><w:rsid w:val="007770B7"/><w:rPr><w:sz w:val="16"/><w:sz-cs w:val="16"/></w:rPr></w:style><w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"
        w:type="paragraph" w:styleId="CommentText"><w:name w:val="annotation text"/><w:basedOn w:val="Normal"/><w:semiHidden/><w:rsid w:val="007770B7"/><w:pPr><w:pStyle w:val="CommentText"/></w:pPr><w:rPr><w:sz w:val="20"/><w:sz-cs
        w:val="20"/></w:rPr></w:style><w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="paragraph" w:styleId="CommentSubject"><w:name w:val="annotation subject"/><w:basedOn
        w:val="CommentText"/><w:next w:val="CommentText"/><w:semiHidden/><w:rsid w:val="007770B7"/><w:pPr><w:pStyle w:val="CommentSubject"/></w:pPr><w:rPr><w:b/><w:b-cs/></w:rPr></w:style>

    </w:styles>

    <w:docPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:displayBackgroundShape/><w:view w:val="print"/><w:zoom w:percent=""/><w:defaultTabStop w:val="709.2887"/><w:docVars/></w:docPr>
    <w:body><w:p xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:pPr><w:pStyle w:val="P1"/></w:pPr><w:r><w:t>.Theinversestrategyofmaximizingnoisewouldnot only find the way back from IBM to Shannon, it may well
    be the only way to enter that body of real numbers originallyknownaschaos</w:t></w:r></w:p><w:sectPr><w:type w:val="next-page"/><w:pgSz xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:w="11907.5525"
    w:h="16839.881" w:orient="portrait"/><w:pgMar xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:top="1133.9978" w:bottom="1133.9978" w:left="1133.9978" w:gutter="0" w:right="1133.9978"/><w:pgBorders
    xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:offset-from="text"/></w:sectPr></w:body>
</w:wordDocument>
```

Illustration 7: XML file created with LibreOffice opened with the texteditor Atom

2 I am writing, the computer is writing

or how to understand software differently

I close all the opened windows by pressing CTRL and “W” repeatedly. I open LibreOffice and start writing – the computer starts writing for me. I am typing this very text and the computer constantly listens, displays and saves.

There are certain trends in software production, that influence our culture towards an increasing gap between sophisticated algorithms and their representation to the user. While the former becomes ever more complicated, the latter is getting more polished with every update. The computational processes get hidden behind user interfaces.
This focus on surfaces, not only reduces software to its interface, but also reflects in the current engagement of art with software. The artistic use of machine learning is a great example of a user that is “stuck” on the interface layer. Instead of engaging with the inner functions of neural networks, artists generate obscure images while mostly talking about datasets, utopia or dystopia (Greene 2018). I do not want to say that these approaches are not valid; While it is important to look at the “superficial” or aesthetic layers of algorithms, this should not obscure the underlying technical processes.
What could this artistic practice with software look like? Software Art provides an interesting example for an art practice that acknowledges the cultural importance and different layers of software at its very core.

Software Art

Software Art describes the »artistic preoccupation with software production« (Cox 2007, p. 147). This means that Software Art is using either the software itself or Code as its material. The subject it addresses are mostly the cultural concepts of software (Cramer 2002b). Software has become so commonplace, that an average user doesn’t even really recognize its existence. A similar effect can be seen in artistic engagement. Software is just part of many digital artworks, not even worth to mention. Software Art instead does not take software for granted and therefore it also realizes how software is made and by whom (Cramer 2002a).
To put focus on the process instead of the end product is not new in the art world, but Software Art exemplifies this approach »appropriate to contemporary conditions« (Cox 2007, p. 147). This enables also to think of software in terms of performance. While the result is not necessarily a fixed product, that is visible, it can be a run-time application, that never reaches the state of finishing. An approach like this opens up new discussions and new ideas. An example of this might be the application »Every Icon« by John F Simon Jr. It is a simple 32 by 32 grid that iterates through every possible combination of black and white squares in the grid. The application has been running since January 14, 1997 and will continue for many years. The application only becomes visible, when you visit the website, which displays the current state. Other then that it performs on its own, reaching formations that will never be seen. In a very neat way this work challenges the viewers imagination about limitations of computation, while automatically producing new, unique images.

By taking software as a primary object of study it acknowledges the role of software in a cultural manner, and realizes that software »is not merely a functional tool, but is itself an artistic creation (nettime.org 2019). This implies also that the code, which software is made of, becomes the material. It means that software is opened up to much more possibilities. Not only art will profit from such engagement, but also the culture around software.

Generative Art
Software Art can be seen as a reaction to the narrow use of software in other fields of digital art for instance Generative Art. In comparison to Software Art the term Generative Art has been around for way longer, following up on Computer Art. But unlike Software Art Generative Art doesn’t consider software as the primary object of study but uses it, if at all, only as a tool. Furthermore Generative Art is mainly focused on the output (Galanter 2003).
Going back to the example of machine learning and the current artistic use. The deep dream is not deep indeed. The use of these algorithms is very flat and mostly concentrates only on the output. Its weird morphed images that are being generated on high-resources machines. And they contain for sure very interesting new ways how to program, but this stays untouched by artists. When the images that we see around as outputs of these algorithms can be considered as Generative Art, how could Software Art be used to create a deeper understanding of this technology? For example it could also be possible to investigate in algorithms, or part of it or narratives around neural networks itself, instead of showing morphed images that happened to come out of pre-written examples. Of course experimentation with such new algorithms should be welcomed and can be helpful to find ways into new territory, but at the same time it is often being forgotten about engaging with the actual software and algorithm that they are using. So the artistic engagement with software should not only regard software as a “pragmatic aid” but carefully look at all the different actors at stake (Arns 2004).

There is a tension between the understanding of Generative Art and Software Art that can be productive and helpful to understand new technologies. First of all this distinction makes obvious how versatile software is being used. Secondly this makes obvious the gap between the surface and the underlying material. It is important to talk about both, how software works and how it is represented. Eventually, it needs to be the aim of both, art and software engineering to find a productive way to talk about these gaps and differences of software function and interface.

The same spiral of dependencies as pointed out in the first part of the work also holds true for creative software. If creativity is dependent on software – and it is, as we use it for almost every artwork or design, this means that software can limit the ways we think. Only if software is questioned in the creation of artworks the creativity can be freed again from this dependency.

Software Studies
Some past publications have dealt with another examination of software, especially in a cultural framework. Software Studies by Matthew Fuller for instance provides a lexicon with diverse objects of software that are being examined. On the blurb of the dust cover of the book Fuller states: »The growing importance of software requires a new kind of cultural theory that can understand the politics of pixels or the poetry of a loop [...].« (Fuller 2008). I think this is an important realization, which opens up the field of software to many different possibilities of understanding and researching.

[works in the field of Software Studies:
* Software Studies, Matthew Fuller
* http://computationalculture.net/ (online journal by Fuller)
* Coding Literacy, Annette Vee
* The Stack, Benjamin H. Bratton
* The Stuff of Bits, Paul Dourish
* Machine Learners, Adrian Mackenzie
* How to be a geek, Matthew Fuller
* Software Theory, Frederica Frabetti
* The Philosophy of Software, David M. Berry
…]

Freeing software
Former artists had to write software to be able to generate digital artworks, nowadays software is widely available, so it is not necessary to engage with it. This of course means a decline in engagement with software and comes with the risk to take software for granted, without questioning it. But the positive consequence is that this frees programming from certain aspects and gives room for a new engagement, “just as previously the invention of photography perhaps freed painting from figurative representation” (Cox 2007, p. 155). This also means that software can and should be used more diverse. I think software can be fun, and therefore software can be revealing and also poetic. Software should be explored like one plays with photography or different materials of painting. Brenda Laurel describes Computers as a theatre, due to the factors of runtime, interaction and space (Laurel 2013). The execution of software can be seen as a performance. When the program is executed the code turns into machine actions. In this sense the metaphor of the theatre is quite precise. Just like in the theatre the script turns into a sequence of actions. Some processes get displayed, some not (e.g. actors getting dressed).

[connect theatre / processbased with idea of poetic software]

»The strong claim for aesthetic computing is that by introducing ideas and methods from art and design into computing, new practices and approaches will emerge responding to new objectives that would not naturally have evolved within the computer sciences and engineering.« (Fishwick 2008, p. 31)

I think that the involvement of art with software can present a useful and contemporary way to change how software is perceived and how we deal with software. The history of Software Art shows that this engagement is possible and revealing. In the following I want to point out why I think art can help in the understanding and use of software with a special focus on the underlying processes that are often hidden in programs.

Software is so complex in its relations and so versatile in its effects, that it might be hard to go about a structured analysis. Instead the arts might provide a field of exploration and experimentation, which can at the same time question and enrich the culture around software. Artistic practice has show that it can occupy fields that are not completely understood, like in the field of music. Art offers the opportunity to deeply engage with certain aspects of software

and connect the cultural to the scientific realm. Also software can be created by artists to express in new ways and comment on different recent developments, as pointed out with different examples previously.
Although this has to be handled in a subtle way, as a wrong approach can also quickly cause misconceptions. I can cause imaginations that are not helpful for the engagement or understanding of technology. The potential to build stories and trigger different imaginations about software or hardware, is a powerful tool to work with as an artist.
Art can bring the hidden layers of software to display in interesting and engaging ways. It is a way of intervening in the production of software but also in the (daily) use of it.

Adorno:
- massproduction of culture industry
- products for a broad audience
- commercial marketing of culture
→ robs peoples imaginations
→ takes over their thinking

Similar case with software
→ Software Art as a counter-action

Conclusion

The observation of software like done above can also be seen as a method for archiving the current state of software.[8] It shows in very detail how I interact with software and how I think about it. Therefore it also states how software is constructed and how it displays to the user. It shows personal relations towards computers and subjectivity in software.

Next to his literary work Kittler left a great amount of software in his estate. People archiving his work where confronted with great problems when trying to preserve the software he wrote (http://traumawien.at/stuff/theory/volume1-feigelfeld.pdf). How can one archive software? If you only save the program code, this bit of code might very quickly become incomprehensible. Computation changes very fast and so do the programming languages. That means that in a very short amount of time certain languages become deprecated and can not be executed anymore. The most present example is Flash. Many interesting works have been created in this language, but due to many different factors Flash is not used anymore. As a consequence many digital artworks can not be executed easily. So you would have to archive whole frameworks or even the whole hardware with the software? This question challenges many factors and might not be solved in a very long time. It shows once again the complexity and the linkages of software.
A different approach on how to archive could well be thought of through art. In an active way, e.g. if artworks deal with the history and the present of software production, it can be a good way to activate and preserve code and its performance. This can happen through its narratives, through its output or subject. Or through writing about how software is used or created, just like this essay attemps to capture the current state of software through detailed observation.

8

I click save and close LibreOffice. The operating system kills the process and shows the empty desktop.

»This two-line BASIC program kills itself. It’s really sad. It makes my cry when I see it.

```
  10 PRINT “Goodbye, cruel world”
  20 NEW
  RUN«
```

(Levin 2002, p. 80)

References

Anon., 2018. The Open Group Base Specifications Issue 7, 2018 edition. [online]. Available from: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/kill.html [Accessed 5 Mar 2019].

Arns, I., 2004. Read_me, run_me, execute_me. *In*: *read_me, Software Art and Cultures*. Aarhus: Aarhus University Press, 120–163.

Bertini, L., 2019. *runme.org - say it with software art!* [online]. Available from: http://runme.org/project/+ViCon/ [Accessed 28 Feb 2019].

Cain, A., 2017. *The Life and Death of Microsoft Clippy, the Paper Clip the World Loved to Hate* [online]. Artsy. Available from: https://www.artsy.net/article/artsy-editorial-life-death-microsoft-clippy-paper-clip-loved-hate [Accessed 28 Feb 2019].

Constant, 2018. *The Techno-Galactic Guide to Software Observation*.

Cox, G., 2007. Generator: The Value of software Art. *In*: Rugg, J. and Sedgwick, M., eds. *Issues in Curating Contemporary Art and Performance*. Intellect Books, 147–162.

Cramer, F., 2002a. *Concepts, Notations, Software, Art* [online]. Available from: http://cramer.pleintekst.nl/essays/concept_notations_software_art/concepts_notations_software_art.html [Accessed 28 Feb 2019].

Cramer, F., 2002b. Contextualizing Software Art, 9.

Davis, F., 2016. We shape our tools and, thereafter, our tools shape us. *Medium* [online]. Available from: https://medium.com/@freddavis/we-shape-our-tools-and-thereafter-our-tools-shape-us-1a564cb87484 [Accessed 27 Feb 2019].

De Decker, K., 2018. *How to Build a Low-tech Website?* [online]. LOW←TECH MAGAZINE. Available from: https://solar.lowtechmagazine.com/2018/09/how-to-build-a-lowtech-website.html [Accessed 27 Feb 2019].

Finn, E., 2017. *What Algorithms Want: Imagination in the Age of Computing*. Cambridge, MA: The MIT Press.

Fishwick, P., 2008. *Aesthetic Computing*. MIT Press Ltd.

Fuller, M., 2008. *Software Studies - A Lexicon*. Cambridge, Mass: The MIT Press.

Galanter, P., 2003. *What is Generative Art? Complexity Theory as a Context for Art Theory* [online]. Available from: https://www.philipgalanter.com/downloads/ga2003_paper.pdf [Accessed 28 Feb 2019].

Greene, T., 2018. *Someone paid $432K for art generated by an open-source neural network* [online]. The Next Web. Available from: https://thenextweb.com/artificial-intelligence/2018/10/25/someone-paid-432k-for-art-generated-by-an-open-source-neural-network/ [Accessed 28 Feb 2019].

Hadler, F., Haupt, J., Andrews, T. L., Callander, A., Flender, K. W., Haensch, K. D., Hartmann, L. F., Hegel, F., Irrgang, D., Jahn, C., Lialina, O., Szydlowski, K., Wirth, S., and Yoran, G. F., 2016. *Interface Critique*. Berlin: Kulturverlag Kadmos Berlin.

Hu, T.-H., 2016. *A Prehistory of the Cloud*. Reprint edition. Cambridge, Massachusetts: MIT Press.

Kitchin, R., 2011. *Code/Space - Software and Everyday Life*. Cambridge, Mass: MIT Press.

Kittler, F., 1992. There is no Software. [online]. Available from: https://monoskop.org/images/f/f9/Kittler_Friedrich_1992_1997_There_Is_No_Software.pdf [Accessed 28 Feb 2019].

Laurel, B., 2013. *Computers as Theatre*. 2nd ed. Upper Saddle River, NJ: Addison-Wesley Professional.

Manovich, L., 1999. Database as Symbolic Form. *Convergence*, 5 (2), 80–99.

Mansoux, A., 2017. Sandbox Culture: A Study of the Application of Free and Open Source Software Licensing Ideas to Art and Cultural Production. [online]. Available from: https://monoskop.org/images/e/ea/Mansoux_Aymeric_Sandbox_Culture_A_Study_of_the_Application_of_FLOSS_Licensing_Ideas_to_Art_and_Cultural_Production_2017.pdf [Accessed 28 Feb 2019].

nettime.org, 2019. *[rohrpost] transmediale.01 newsletter #3a - Conference 8 Feb: Software* [online]. Available from: http://amsterdam.nettime.org/Lists-Archives/rohrpost-0101/msg00039.html [Accessed 28 Feb 2019].

Parks, L., 2009. Around the Antenna Tree: The Politics of Infrastructural Visibility. [online]. Available from: http://www.flowjournal.org/2009/03/around-the-antenna-tree-the-politics-of-infrastructural-visibilitylisa-parks-uc-santa-barbara/ [Accessed 28 Feb 2019].

Reas, C., 2019. *{Software} Structures by Casey Reas et al.* [online]. Available from: https://artport.whitney.org/commissions/softwarestructures/text.html [Accessed 28 Feb 2019].

Reeves, B. and Nass, C., 2003. *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Reprint. Stanford, Calif: Center for the Study of Language and Inf.

Tenen, D., 2017. *Plain Text: The Poetics of Computation*. Stanford University Press.

Ullman, E., 2012. *Close to the Machine*. Reprint edition. New York: Picador Paper.

Wardrip-Fruin, N., 2012. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. Cambridge, Mass London: The MIT Press.

Winograd, T. A. and Flores, F., 1995. *Understanding Computers and Cognition: A New Foundation for Design*. New ed. Boston: Addison Wesley.

Yuill, S., 2004. Code Art Brutalism: Low-Level Systems and Simple Programs. *In*: *read_me, Software Art and Cultures*. Aarhus: Aarhus University Press, 120–163.

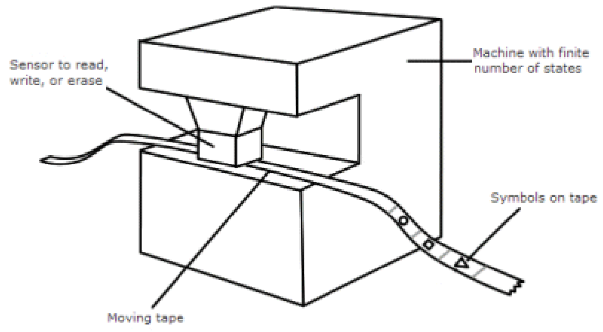
Read/Write (a poem about reading and writing software)

I am reading while I am writing. Without reading I cannot write.
The act of writing on a computer requires looking at the keyboard and on the screen at the same time.
The computer is writing for me.
Typing on the screen means that the computer is writing every typed letter to the memory.
I am typing.
The act of writing on a computer requires an input device. The fingers are typing.
I am reading.
The computer shows pixels on the screen which form letters and words.
The computer is reading. Without reading the computer cannot write.
The computer is reading the input of the keyboard and the bits that are stored inside the memory (Random Access Memory)
I am writing code.
As soon as the writing shaped according to semantic rules, that can be understood by a compiler, the text that is written is also code. The writing of code is based on software itself.
The computer is writing code.
The computer is again writing what I have written to its memory. Contemporary computers even literally automatically write parts of the code, as they are programmed to autofill gaps.

I am reading code.
I am reading what I have written to see if the syntax is correct.
You are reading my code.
We do not only write code for ourselfs but also for other people.
The computer is reading my code.
The computer is writing my code to its memory.
Your computer is reading my code.
The computer needs to read and write in order to display.
I am compiling.
Your computer is writing my code.
The computer is reading and writing.
You are writing.
The process of compiling requires the computer to read the syntax of the code and write machine instructions in the shape of a executable file.
I am clicking.
Your computer is writing.
The click on the executable file launches the program.
The computer is executing.
Your computer is reading.
The moment when the computer is following the instructions in the code in form of current flows.

Therefore the computer is reading and writing.
This means that the computer is again reading the instructions and writing through the code instructions.

A Turing machine is a theoretical generalized computer, composed of a tape on which symbols representing instructions are imprinted. The tape can move backwards and forwards in the machine, which can read the intructions and write the resultant output back onto the tape.


Sensor to read, write, or erase
Machine with finite number of states
Symbols on tape
Moving tape

Software zoo